# Large scale plane wave pseudopotential density functional calculations on GPU clusters


**Long Wang[1], Weile Jia[1], Xuebin Chi[1], Weiguo Gao[2], <u>Lin-Wang Wang[3]</u>**

*(1) Supercomputing Center, Chinese Academy of Science*

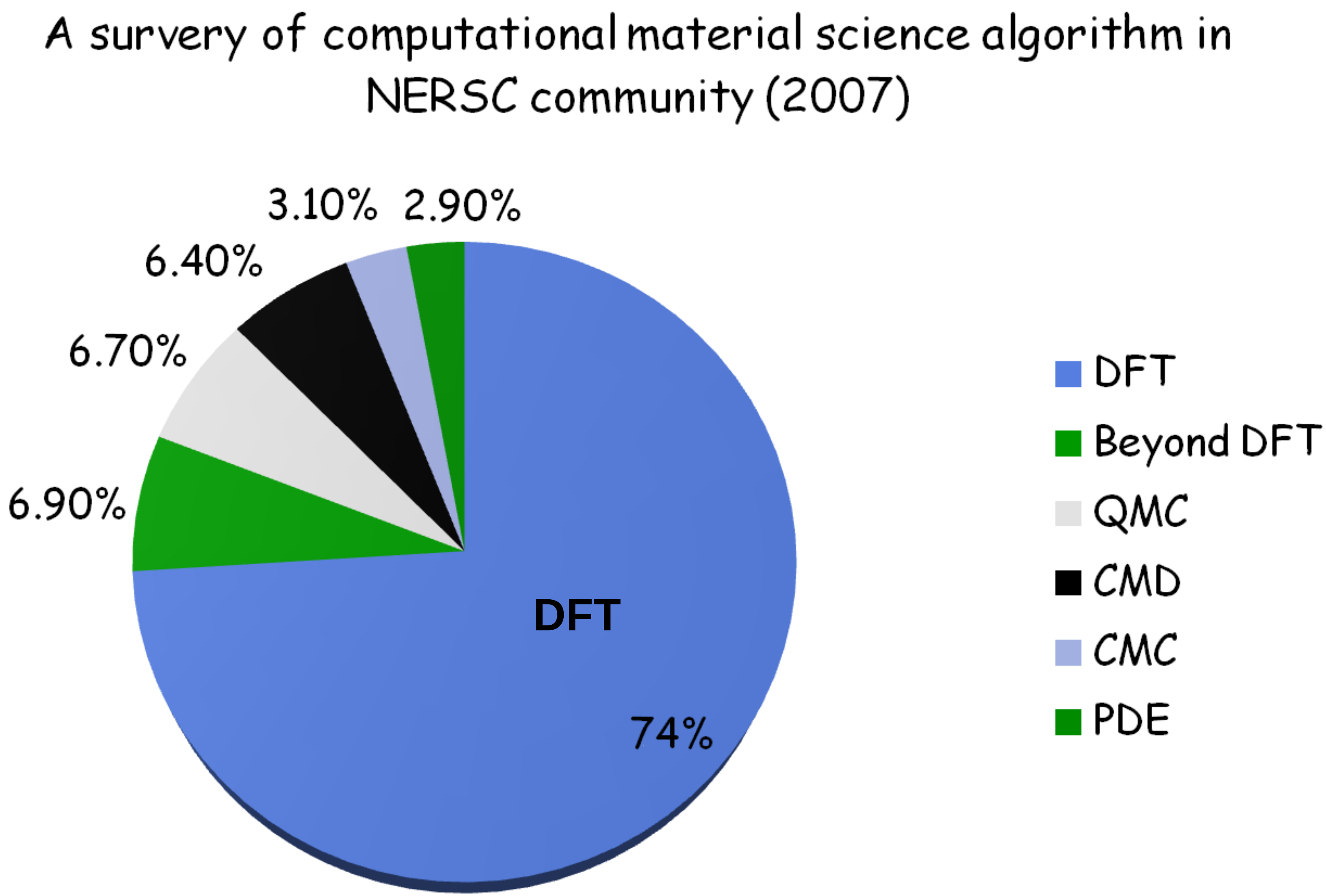
*(2) Fudan University*

*(3) Material Science Division*
*Lawrence Berkeley National Laboratory*



**National Basic Research Program of China**
**NSF of China**
**Science & Technology Commission of Shanghai**
**Office of Science, BES, DOE, USA**

# A profile for material science simulation



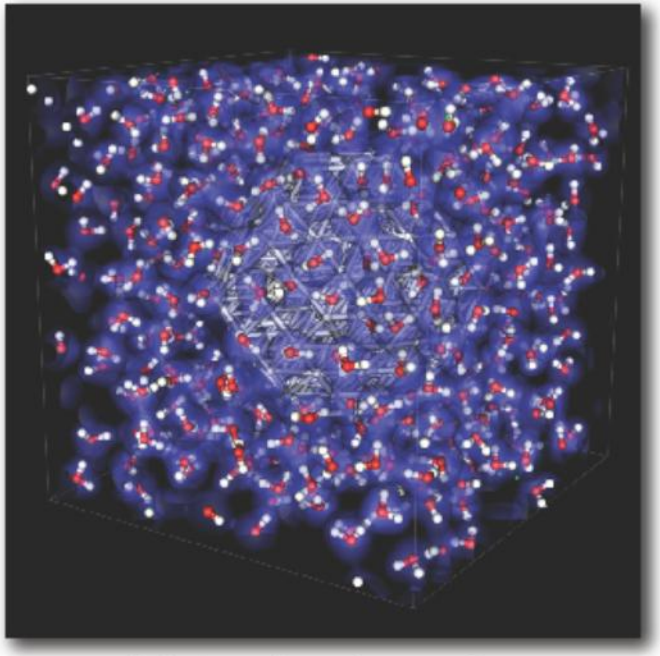
# What is the remaining challenge for DFT calculations?

- ❖ 100 to 1000 atoms
- ❖ Ab initio MD for a few ns
- ❖ massive configuration space search for structures

State-of-the-art: 1-2 min per MD step
(so can only calculate a few ps,
But want: ns!)

For >>1000 atoms, linear scaling method

## Nanocatalysis: Pt



Molecular dynamics
$Pt_{201}$+427$H_2O$ 1482 atoms

P. Kent, ORNL
M. Neurock, U. Virginia

**Sweet spot: a few hundreds to a few thousand atoms need faster speed**

# Plane Wave Pseudopotential DFT codes



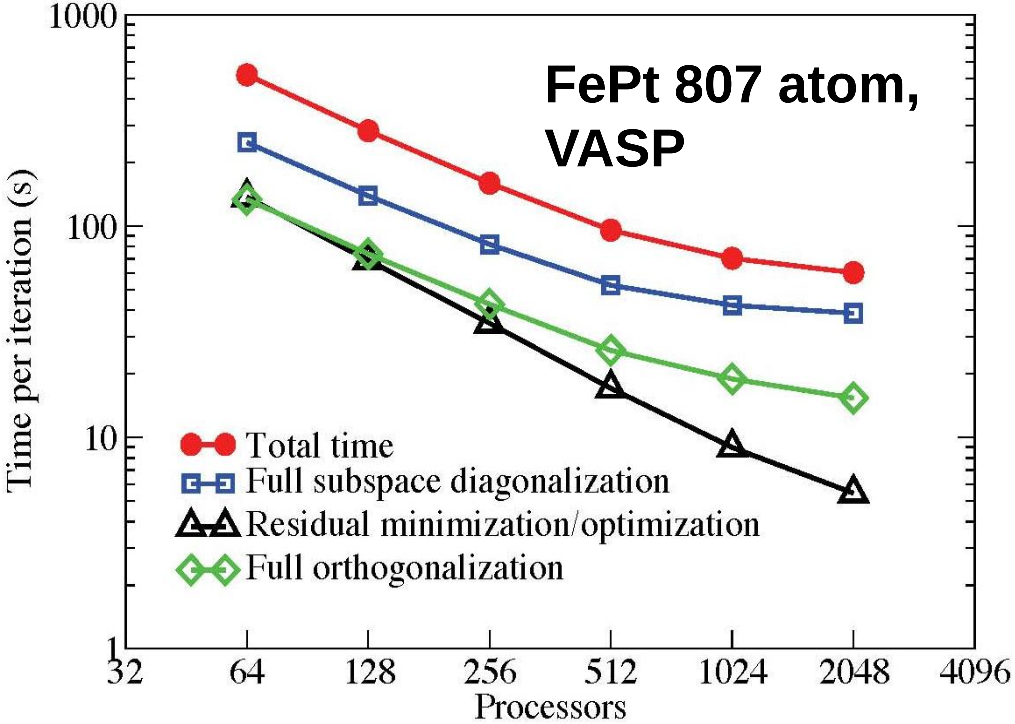
- They are the most widely used, and most mature codes

- There are about a dozen of them:
  VASP, CASTEP, CPMD, ABINIT, PWSCF, DACAPO, SOCORRO, DFT++, PARATEC, DOD-PW, CP2K, SPHINX, QBOX, PEtot

- But the CPU codes often do not scale (e.g., 1000 atom system might scale to a few thousand cores)

- A few minutes per MD step

**Idea: use GPU to speed up the absolute speed**




P. Kent, ORNL

# The computational cost of DFT method

$$[-\frac{1}{2}\nabla^2 + V_{tot}(r)]\psi_i(r) = \varepsilon_i \psi_i(r)$$

- If the size of the system is *N*:
- *N* coefficients to describe one wavefunction $\psi_i(r)$
- $i = 1,\ldots, M$ wavefunctions $\psi_i(r)$, *M* is proportional to *N*.
- Orthogonalization: $\int \psi_i(r)\psi_j^*(r)d^3r$, $M^2$ wave function pairs, each with *N* coefficients: $N*M^2$, i.e $N^3$ scaling.

The repeated calculation of these orthogonal wave functions make the computation expensive, $O(N^3)$.

# PEtot code

- Developed in Lawrence Berkeley National Lab
- Free: https//hpcrd.lbl.gov/~linwang/PEtot/PEtot.html
- Has three levels of parallelization: G-space, state index, k-point
- Uses norm conserving pseudopotential and ultra-soft psd.
- Use parallel FFT (by Andrew Canning)
- Can calculate 10,000 states on a few thousand processors

# The flow chart of the DFT method (PEtot code)

## The overall flow chart of SCF iterations

$$\{-\frac{1}{2}\nabla^2 + V(r) + \sum_l |\phi_l\rangle\langle\phi_l|\}\psi_i(r) \equiv H\psi_i(r) = E_i\psi_i(r)$$

$$\{\psi_i\}_{i=1,\ldots,N}$$

$$\rho(r) = \sum_i^N |\psi_i(r)|^2$$

$$V(r)$$

Selfconsistency

## The conjugate-gradient (CG) to solve the Schrodinger's eq (98% of the total time)

$$h(i,j) = \langle\psi_i|H|\psi_j\rangle$$ Sub_diag, *

$$P_i = H\psi_i - \varepsilon_i\psi_i$$ Hpsi, *

$$P_i = A(P_i - \frac{\lambda_i}{\lambda_i^o} P_i^o)$$ Precond. CG step

$$P_i = P_i - \sum_{j=1,i} \langle P_i|\psi_j\rangle$$ Projection, *

$$\psi_i = \psi_i \cos\theta_i + P_i \sin\theta_i$$ Line minimiz.

$$\psi_i = \psi_i - \sum_{j<i} \langle\psi_i|\psi_j\rangle$$ Orth., *

nline

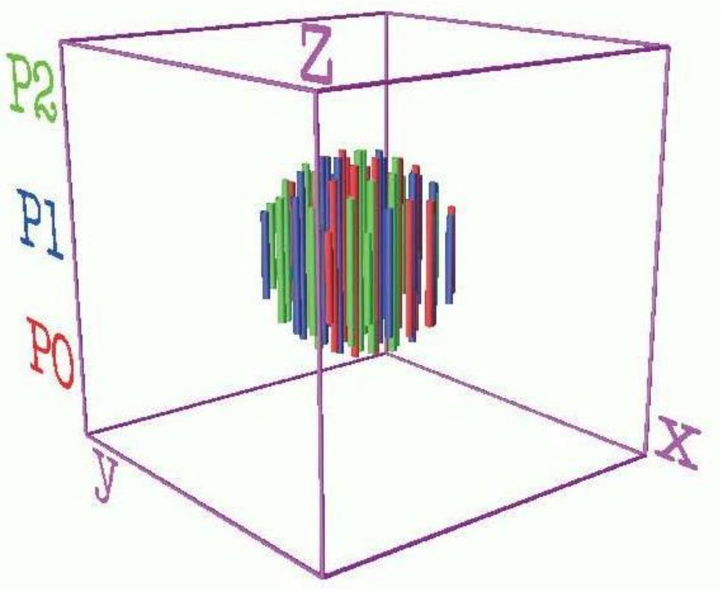
$$h(i,j) = \langle\psi_i|H|\psi_j\rangle$$ Sub_diag, *

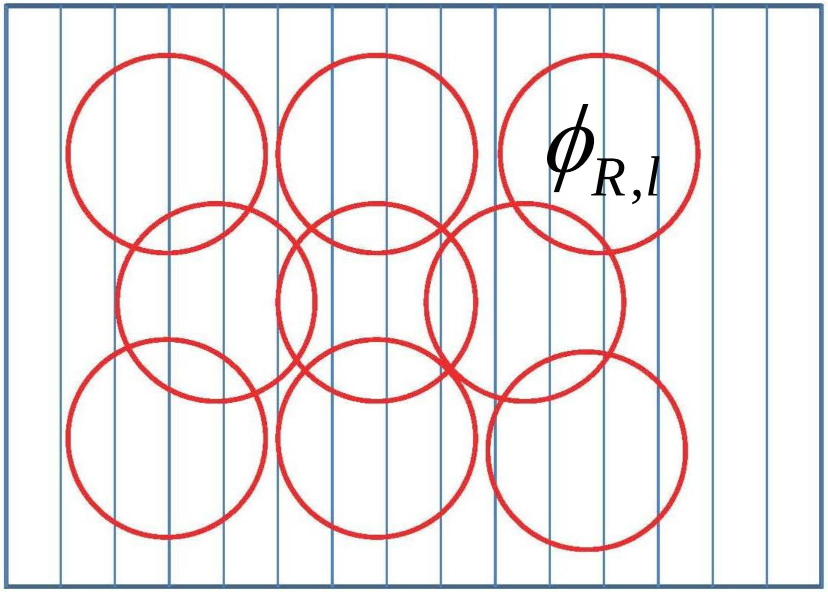
# The kernels in the H*ψ  (Hpsi)

$$\{-\frac{1}{2}\nabla^2 + V(r) + \sum_l |\phi_l\rangle\langle\phi_l|\}\psi_i(r)$$

FFT (by A. Canning)

Real sace
Nonlocal pseudopotential




$$\sum_{R,l} |\phi_{R,l}\rangle\langle\phi_{R,l}|\psi_i\rangle$$

# Parallelization scheme for a CPU code

$$\psi_{i,k}(r)=\sum_{G} C_{i,k}(G)\exp(-i(G+k)\bullet r)$$

**$k_1$**

| | | | | |
|---|---|---|---|---|
| $P_{00}$ | $P_{01}$ | $P_{02}$ | $P_{03}$ | $G_1$ |
| $P_{10}$ | $P_{11}$ | $P_{12}$ | $P_{13}$ | $G_2$ |
| $P_{20}$ | $P_{21}$ | $P_{22}$ | $P_{23}$ | $G_3$ |
| $P_{30}$ | $P_{31}$ | $P_{32}$ | $P_{33}$ | $G_4$ |
| $\Psi_1$ | $\Psi_2$ | $\Psi_3$ | $\Psi_4$ | |

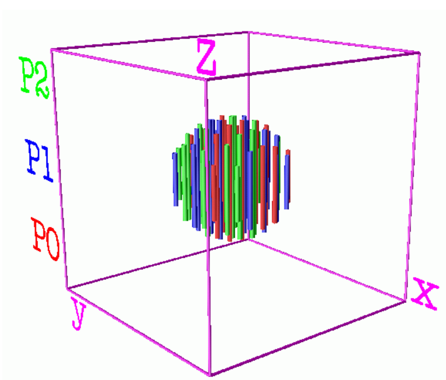
……..

**$k_n$**

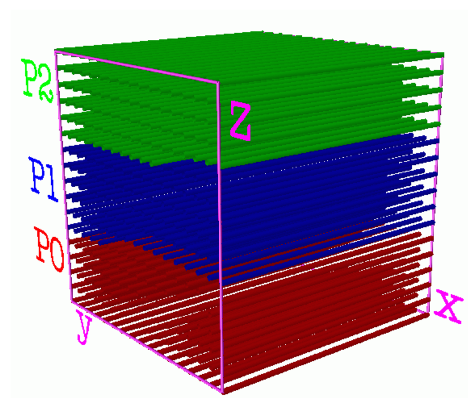
| | | | | |
|---|---|---|---|---|
| $P_{00}$ | $P_{01}$ | $P_{02}$ | $P_{03}$ | $G_1$ |
| $P_{10}$ | $P_{11}$ | $P_{12}$ | $P_{13}$ | $G_2$ |
| $P_{20}$ | $P_{21}$ | $P_{22}$ | $P_{23}$ | $G_3$ |
| $P_{30}$ | $P_{31}$ | $P_{32}$ | $P_{33}$ | $G_4$ |
| $\Psi_1$ | $\Psi_2$ | $\Psi_3$ | $\Psi_4$ | |

**Parallel FFT**

**(each CPU has many 1D FFTs)**




**$G_1, G_2, G_3$ (G-space)**




**Real space**

# GPU hybrid parallelization

## G-parallel

$G_0$ $P_0$

.

.

$G_{14}$ $P_{14}$

$G_{15}$ $P_{15}$

$\{\psi_i\}$

$$\langle \psi_i | \psi_j \rangle$$

**Diag**
**rotation**

**CUBLAS**
**MPI_allreduce**

**Wave function transpose**

**MPI_alltoall**

## Index parallel

$P_0$ . . . . $P_{14}$ $P_{15}$ $\{G\}$

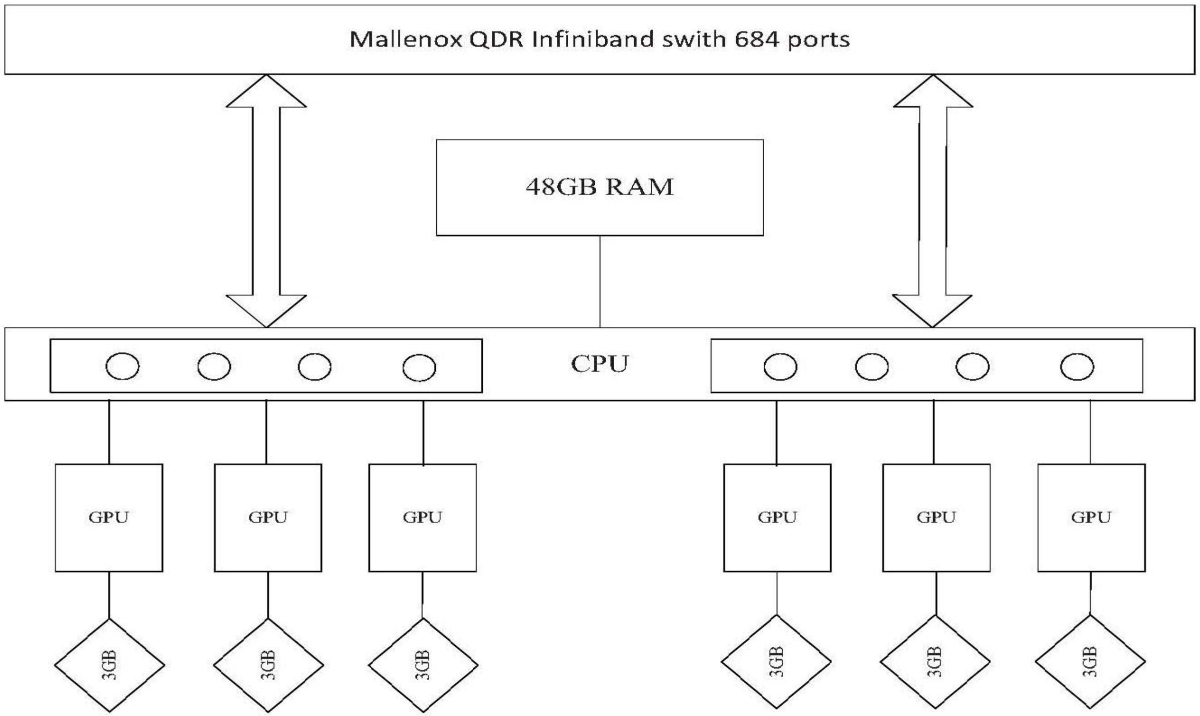
$\Psi_0$ $\Psi_{14}$ $\Psi_{15}$

**Hpsi**
**FFT**
**nonlocal**

**CUFFT**

- **The FFT is within a single GPU (no parallel FFT)**
- **memory limitation to the size: a few thousand atoms**

# A single node in the CPU/GPU machine (IPE)



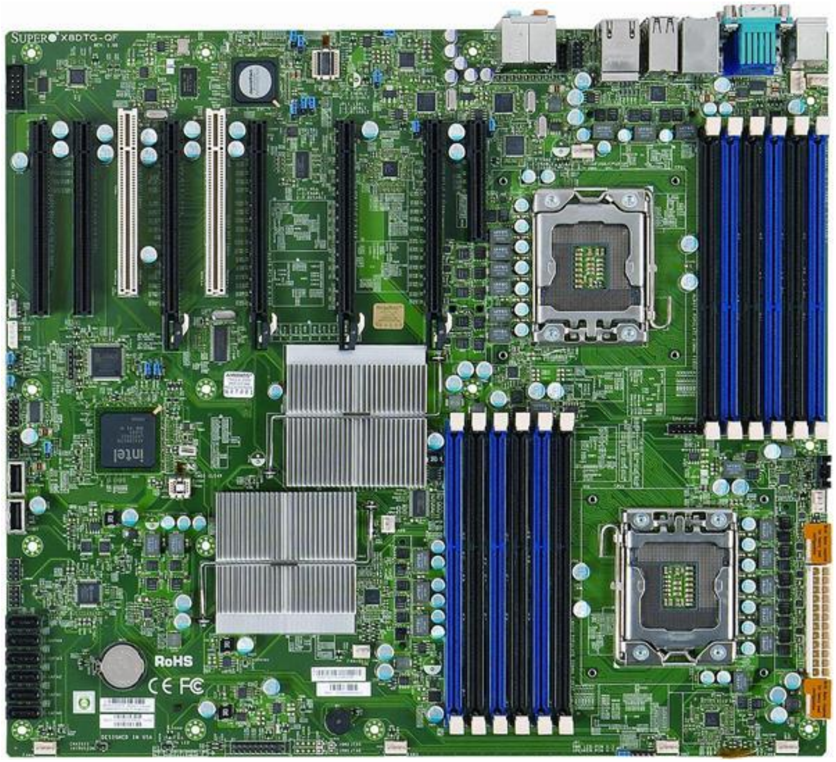

**CPU : Xeon 5520 quad-core CPU**
**9 Gflops/core (2.2 GHz)**
**6 GB memory/core**

**GPU: Nvidia Fermi C2050 GPU card**
**448 stream processors/card**
**515 Gflops/card (double precision)**
**3 GB memory/card**

**Multiple GPU cards in one node**
**(Institute of Processing Engineering, CAS)**

**Strategy: one CPU core controls one GPU card, CPU/GPU unit**

# Another example of multiple GPU per node machine

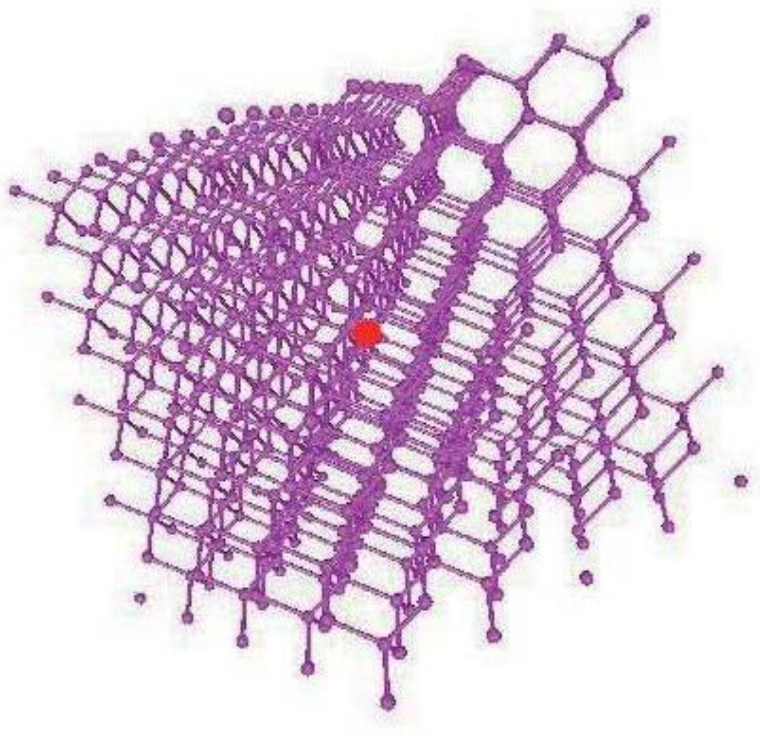



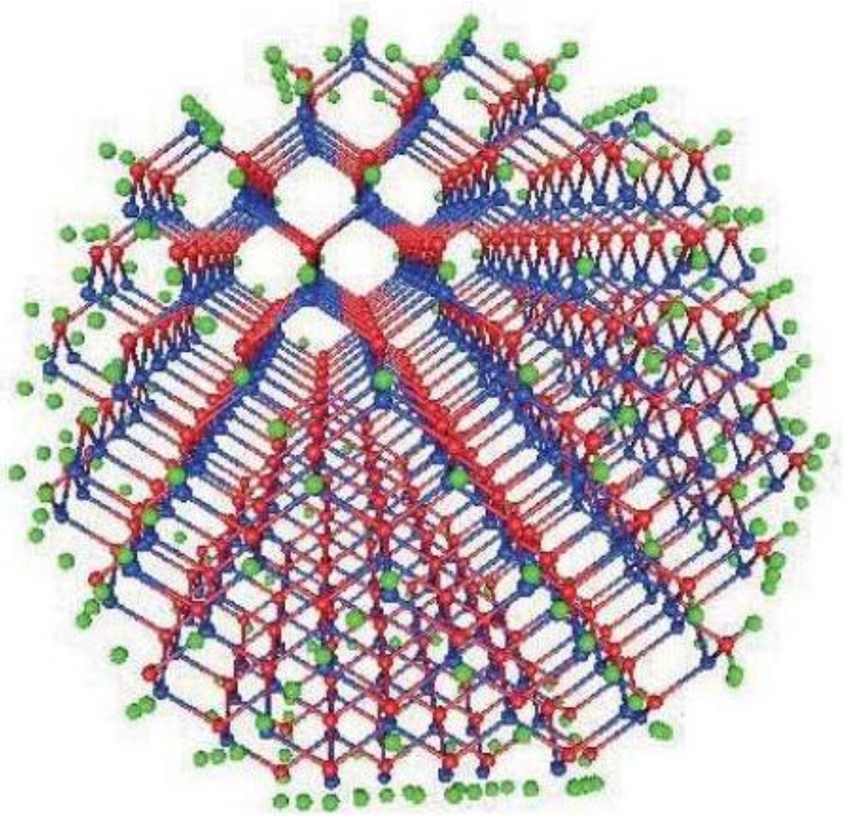

- ❖ NEWTON, offered by Electronics Nexus
- ❖ 8 CPU cores (Intel)
- ❖ 8 GPU cards (Nvidia)
- ❖ Start from $2,199 !

## The testing systems

GaAs:N (512 atoms)
2048 electrons
$128^3$ FFT grid
40 Ryd Ecut
3.3 x$10^5$ PW coeff

CdSe quantum dot (933 atoms)
2832 electrons
$256^3$ FFT grid
30 Ryd Ecut
1.1x$10^6$ PW coeff





# GPU coding (easy to use CUBLAS)

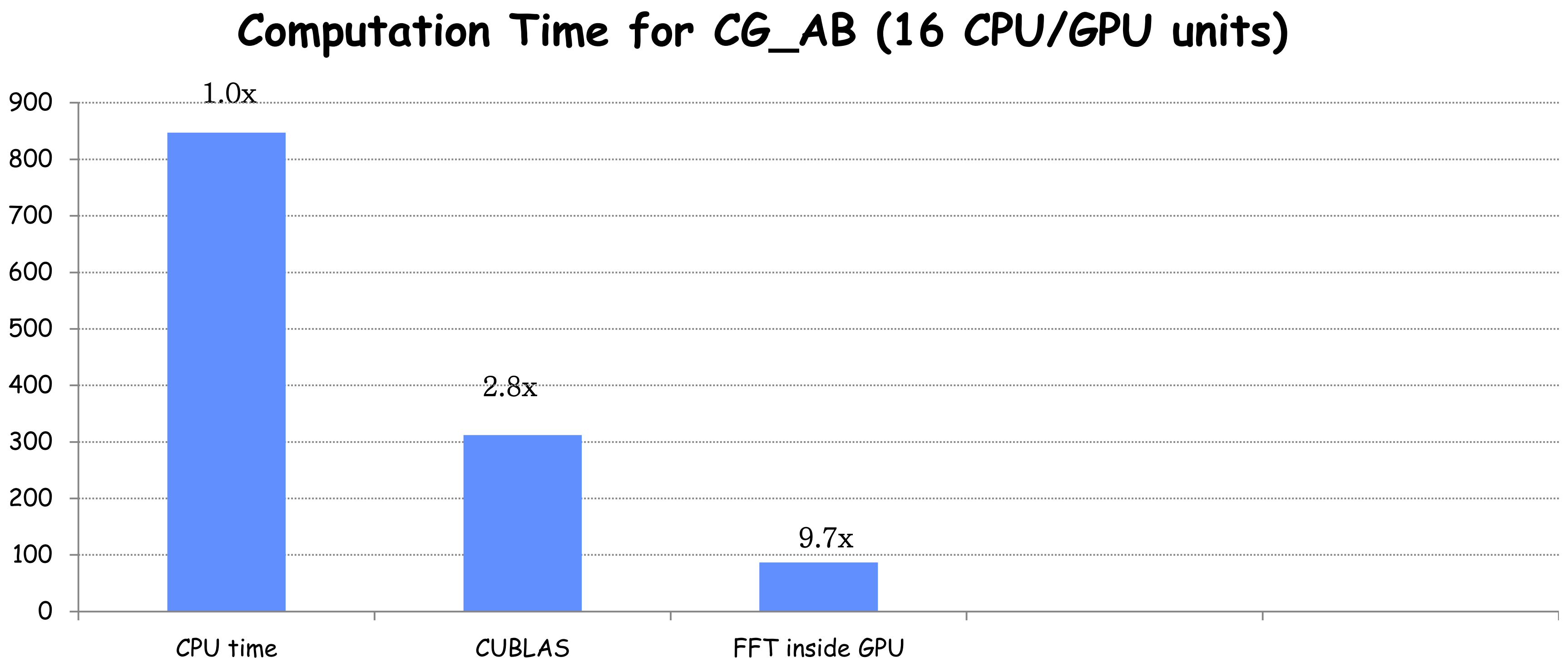
```
CALL zgemm('c','n',mx, mx,ng_n,one,A,mg,B,mg, zero,SS, mx)
```

**CPU code**

```
    stat = cublas_alloc(mg*mx, 16, cu_A)                                    ! Alloc CUDA memory
    stat = cublas_alloc(mx*mx, 16, cu_SS)
    stat = cublas_alloc(mg*mx, 16, cu_B)
call cublas_set_matrix (mg, mx, 16,  A, mg, cu_A, mg)                       ! Copy matrix to GPU
call cublas_set_matrix (mg, mx, 16,  B, mg, cu_B, mg)
call cublas_zgemm('c','n',mx,mx,ng_n,one,cu_A,mg, cu_B,mg, zero,cu_SS,mx)   ! Cublas call
call cublas_get_matrix (mx, mx, 16, cu_SS, mx, SS, mx)                      !  Get matrix to CPU
    call cublas_free(cu_A)
    call cublas_free(cu_B)
    call cublas_free(cu_SS)                                                 ! Free CUDA memory
```

**GPU code**

# Different steps of speeding up to go to GPU



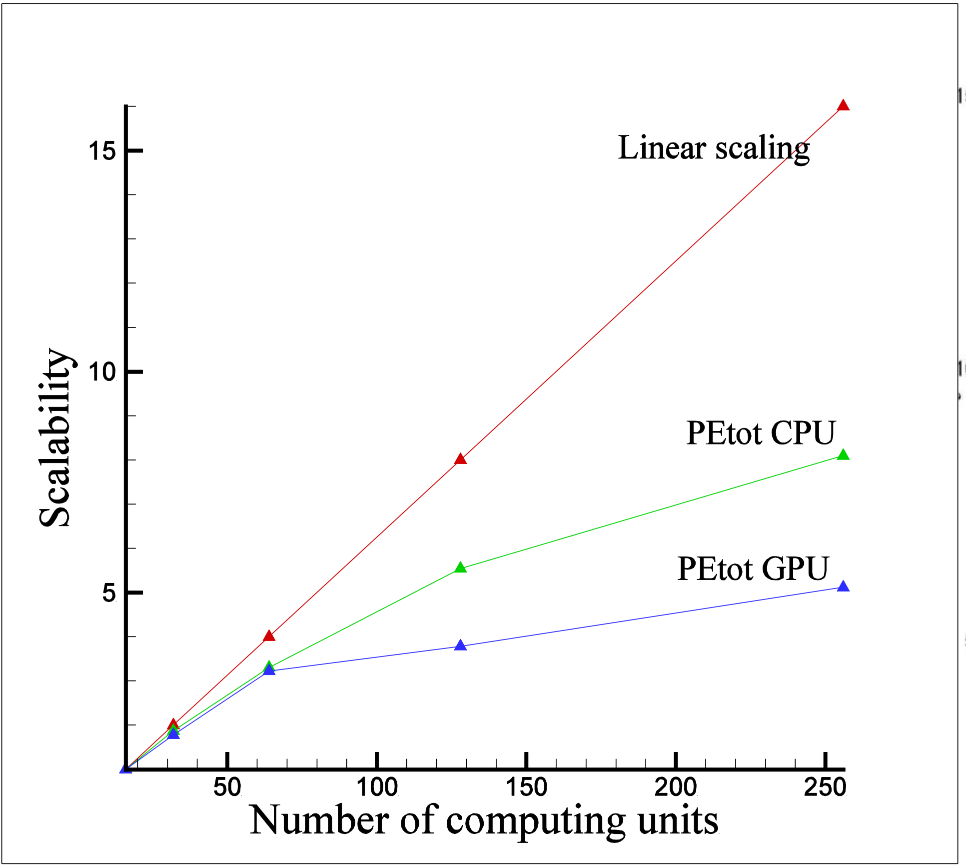
# The results

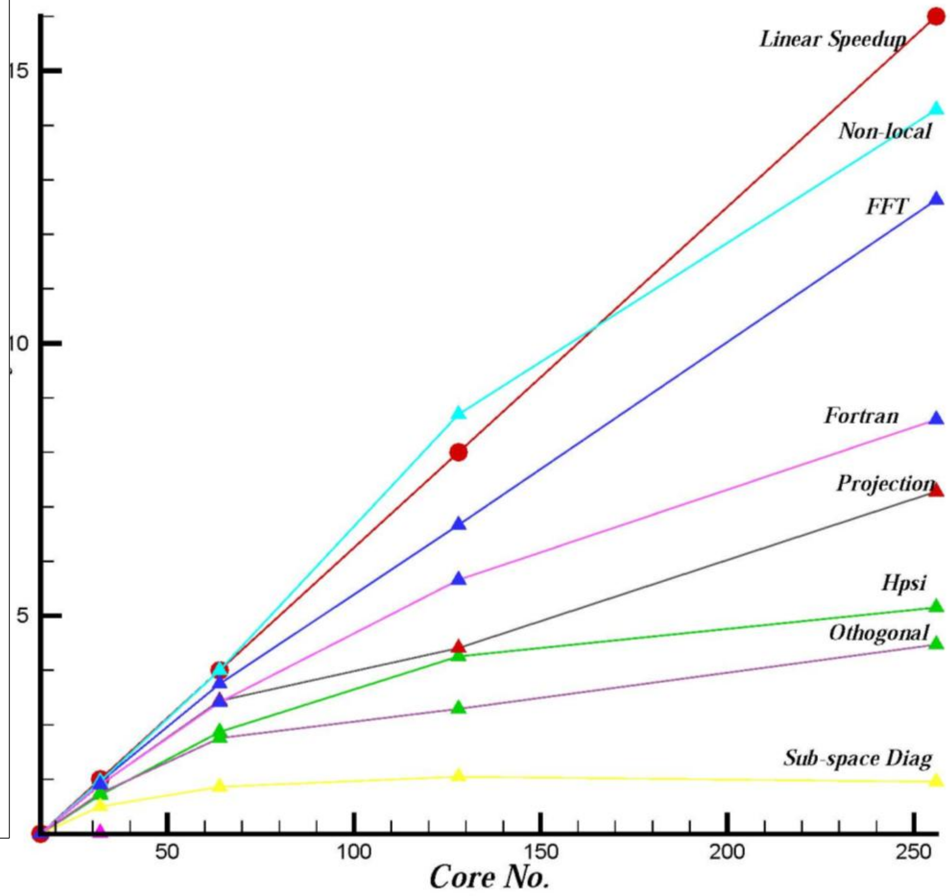


| Computing units | 16 | 32 | 64 | 128 | 256 | 256 |
|---|---|---|---|---|---|---|
| systems | 512-GaAs | 512-GaAs | 512-GaAs | 512-GaAs | 512-GaAs | 933-CdSe |
| PEtot (CPU) | 842 | 450 | 255 | 152 | 104 | 495 |
| PEtot (GPU) | 87 | 49 | 27 | 23 | 17 | 56 |
| Speed-up (PEtot) | 9.7x | 9.2x | 9.4x | 7x | 6.1x | 8.8x |
| Total flops (Tflops) | 0.59 | 1.05 | 1.91 | 2.24 | 3.03 | 5.92 |
| Efficiency | 7.1% | 6.3% | 5.7% | 3.3% | 2.3% | 4.4% |

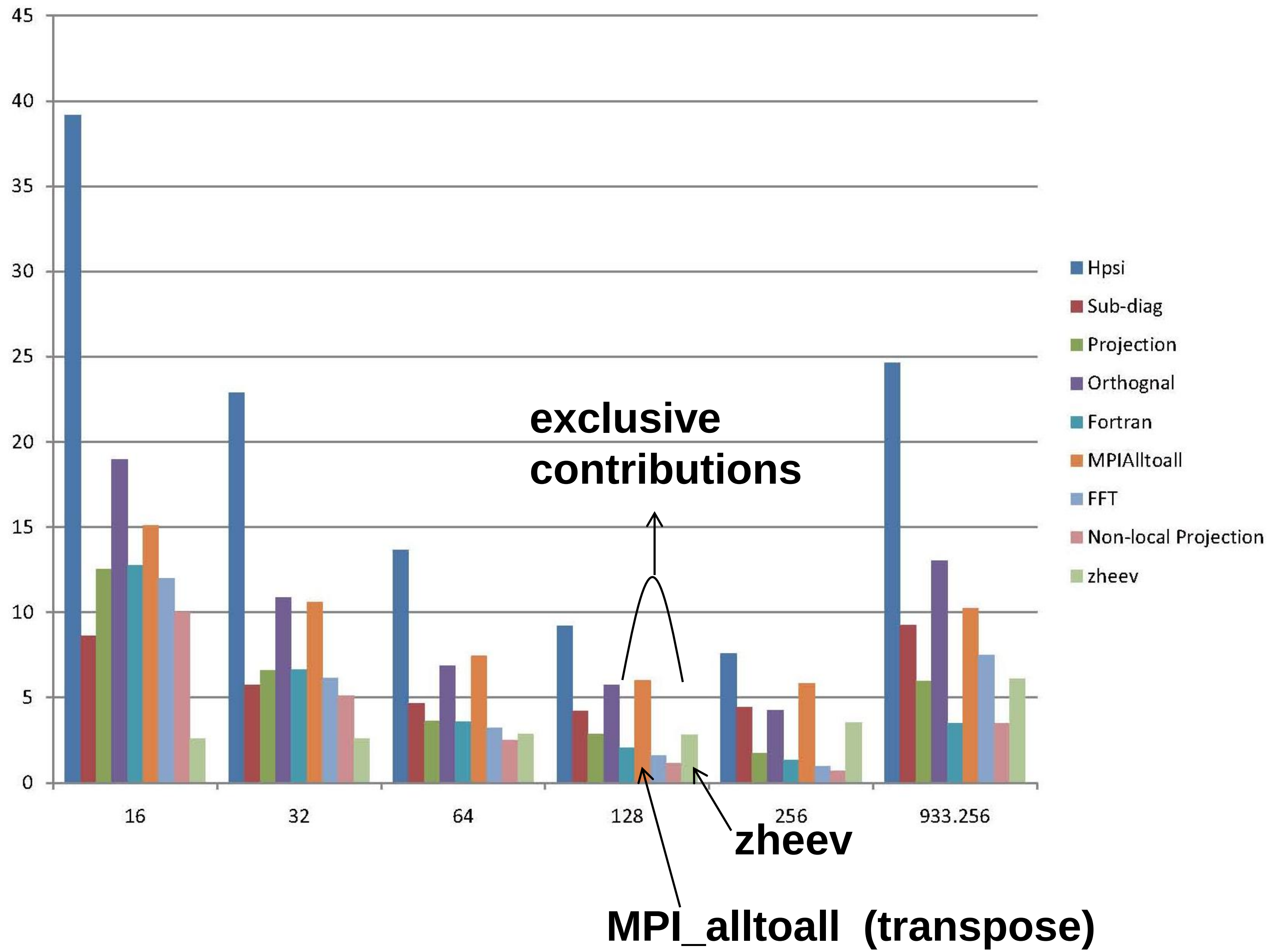
**Computing unit: one CPU core/ one GPU card**
**Times: in seconds**
**4 line min steps in CG_AB**
**Only the CG_AB times are reported**

# The processor scalings



# The total computational times for different kernels



# The remaining problems & solutions

❖ The MPI_alltoall (for transpose) takes time

For P=Hψ-εψ and H*P, reduce the double precision to 4 byte number, hence reduce the MPI_alltoall

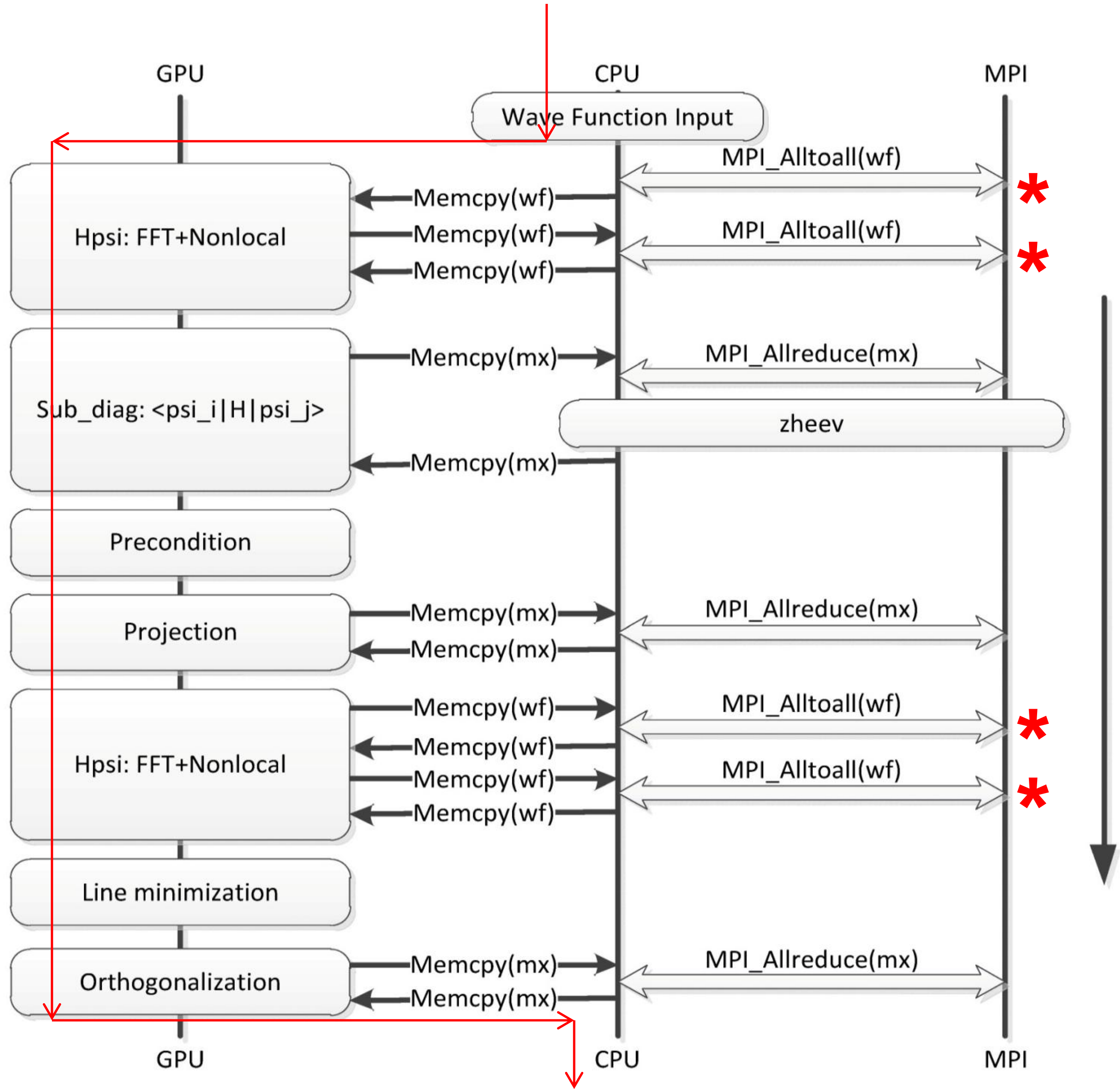
❖ The matrix diagonalization routines take time

Using new CPU and GPU routines for diagonalizations

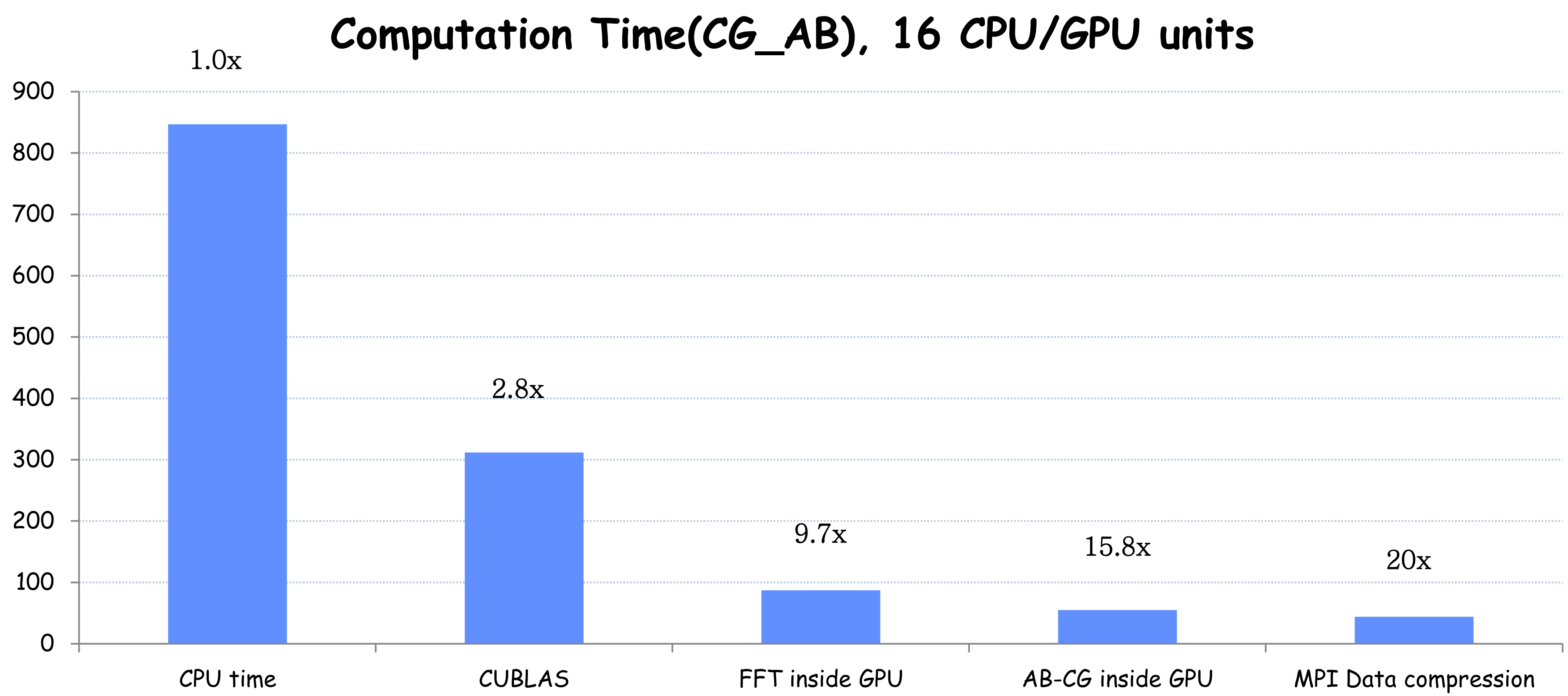
❖ The CPU-GPU wave function data copies take time

Move all the computations to GPU, reduce CPU-GPU data copy

# The new program flow chart



# Different steps of speeding up to go to GPU



# CONCLUSIONS



- It is possible to use GPU to speed up PW Pseudopotential DFT code by x20.
- Need to change the parallelization scheme, and introduce new algorithm.
- Hpsi and FFT are done within one GPU
- Want as many GPU per node as possible, CPU not used
- Want large GPU global memory (one whole wave function will be stored in one GPU)
- Want faster MPI_alltoall, MPI_allreduce
- Want faster GPU multi-processor lib